Lesson 5
Understanding Desktop Applications

Learning Objectives

Students will learn about:

- Windows Forms Applications
- Console-Based Applications
- Windows Services

OBJECTIVE1: Understanding Windows Forms Applications

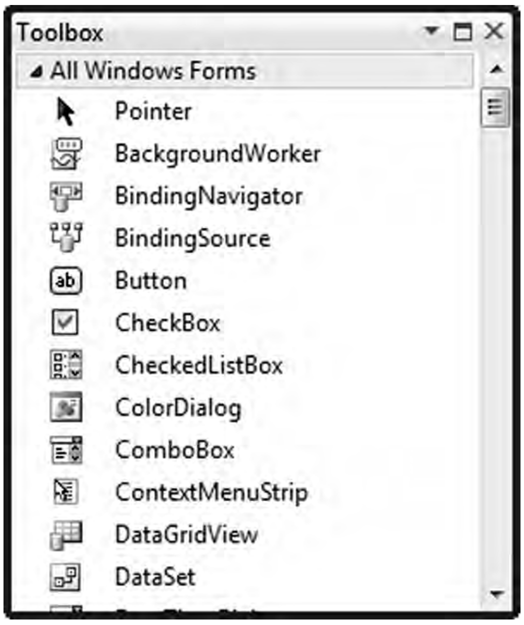
Windows Forms applications are smart client applications consisting of one or more forms that display a visual interface to the user. These applications integrate well with the operating system, use connected devices, and can work whether connected to the Internet or not

Designing a Windows Form

A Windows Form is a visual surface capable of displaying a variety of controls, including text boxes, buttons, and menus. Visual Studio provides a drag-and-drop Windows Forms designer that you can use to easily create your applications.
To design Windows Forms, you must first decide what controls you want to place on the form. Windows Forms provides a large collection of common controls that you can readily use to create an excellent user interface. If the functionality you are looking for is not already available as a common control, you have the option to either create a custom control yourself or buy a control from a third-party vendor.

You can use the functionality provided by Visual Studio's Windows Forms Designer to quickly place and arrange controls per your requirements. Visual Studio provides easy access to the available controls through its toolbox, as shown in Figure 5-1.

Figure5-1


Toolbox
All Windows Forms
Pointer
BackgroundWorker
BindingNavigator
BindingSource
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
ContextMenuStrip
DataGridView
DataSet

TakeNote

A control is a distinct user interface element that accepts input from the user or displays out- put to the user.

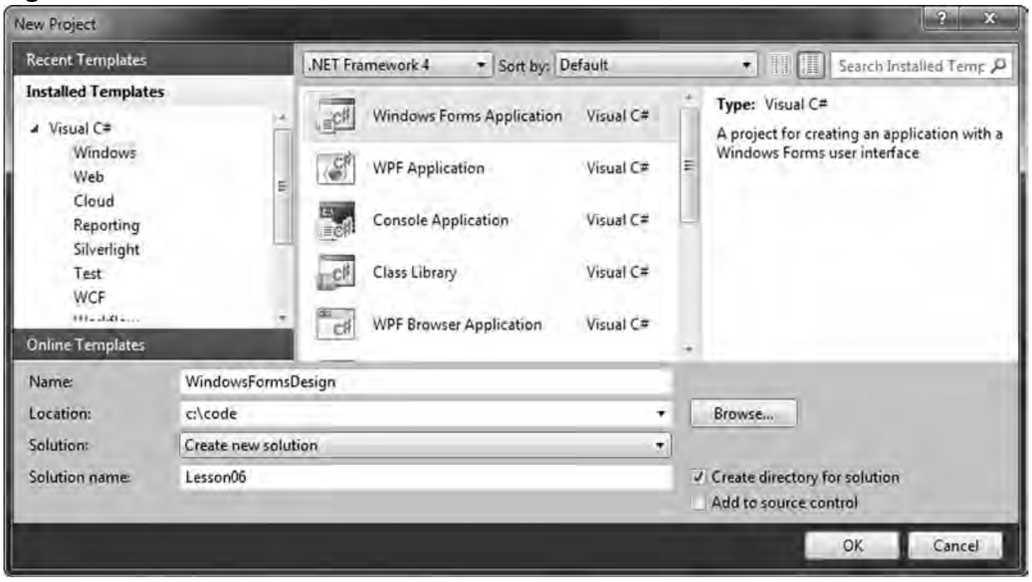
A form and its components generally respond to user actions such as keystrokes or mouse movement. These actions are called events. Much of the code that you write as a Windows Forms developer is directed toward capturing such events and handling them by creating an appropriate response. For instance, in the following exercise, you will create a Windows Form that displays the date value selected by the user

Illustration1: CREATE A WINDOWS FORM

GET READY. Launch Microsoft Visual Studio. Then, perform the following actions:

1. Create a new project based on the Windows Forms Application template, as shown in Figure 5-2. Name the project WindowsFormsDesign

Figure5-2



Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: WindowsFormsDesign

Location: c:\code

Solution: Create new solution

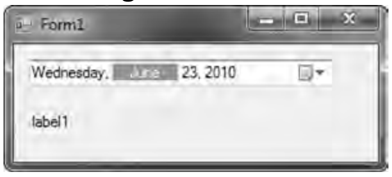
Solution name: Lesson06

☑ Create directory for solution

☐ Add to source control

2. The Windows Form Application project will load with a default form (Form1.cs) opened in the Designer view. The Designer view allows you to work visually with the form. For example, you can arrange controls on the form's surface and set their properties. Available controls can be accessed from the Toolbox window. If you don't already see the Toolbox window, select View > Toolbox to display it. Then, from the Toolbox, drag and drop a DateTimePicker control and a Label control on the Designer surface and arrange the controls as shown in Figure 5-3.

Figure5-3



3. In the Designer view, select the Label control and, using the Properties window, set its Text property to an empty string.
4. Also in the Designer view, double-click the DateTimePicker control. This action attaches the default event handler for the ValueChanged event of the DateTimePicker control and switches the view from Designer to Code. Next, change the default code for the event handler as follows:

```
private void dateTimePicker1_ValueChanged (object sender,
EventArgs e)
{
label1.Text =
dateTimePicker1.Value.ToLongDateString();
}
```

5. Select Debug > Start Debugging (or press F5) to run the project. On the user interface, select a new date and verify that the selected date is displayed on the Label control

In this exercise, note that when the form is initially displayed, the Label control is set to an empty string. Then, as soon as you change the date selection by manipulating the DateTimePicker control, the selected date value is set as the text for the Label control

TakeNote

In this exercise, we used the default control names. In complex forms with more controls, it's always a good idea to give the controls more meaningful names.

Understanding the Windows Form Event Model

Event handling plays a key role in user interface-based programming; through event handling, you respond to various events that are fired as a result of user actions and thus make programs interactive. The Windows Forms event model uses .NET Framework ***delegates*** to bind events to their respective event handlers.

In Windows Forms applications, each form and control exposes a predefined set of events. When an event occurs, the code in the associated event handler is invoked. For instance, in the previous exercise, when you double-clicked the DateTimePicker control to add code to the event handler, Visual Studio generated the following code to attach the event handler to the event

```
this.dateTimePicker1.ValueChanged += new
    System.EventHandler(
    this.dateTimePicker1_ValueChanged);
```

Here, ValueChanged is the event of the DateTimePicker control that we want to capture. So, a new instance of the delegate of type EventHandler is created and the method dateTimePicker1_ ValueChanged is passed to the event handler. The dateTimePicker1_ValueChanged method is the method in which you will actually write the event-handling code.

This code is automatically generated by the Visual Studio Designer. You will find this code in the code-behind file for the designer (Form1.Designer.cs), inside a code region entitled Windows Form Designer generated code.

Yet another thing to notice is that the syntax for adding a delegate uses the += operator. That's because the .NET Framework supports multicast delegates in which a delegate can be bound to more than one method, thus allowing one-to-many notifications when an event is fired.

TakeNote

A delegate can be bound to any method whose signature matches that of the event handler

Using Visual Inheritance

Visual inheritance allows you to reuse existing functionality and layout for Windows Forms. One of the core principles of object-oriented programming is inheritance. When a class inherits from a base class, it derives its base functionality from the base class. Of course, you can always extend the derived class to provide additional functionality and be more useful.

A Windows Form, at its core, is just another class; therefore, inheritance applies to it as well. However, when inheritance is applied to a Windows Form, it causes the inheritance of all the visual characteristics of a form, such as size, color, and any controls placed on the form. You can also visually manipulate any of the properties that are inherited from the base class. Therefore, inheriting Windows Forms is often called ***visual inheritance***.

In the following exercise, you will create a Windows Form via visual inheritance of an existing form

Illustration2: CREATE A WINDOW FORM USING VISUAL INHERITANCE

GET READY. Launch Microsoft Visual Studio and open the existing Windows Application Project named WindowsFormsDesign. Then, perform these steps:

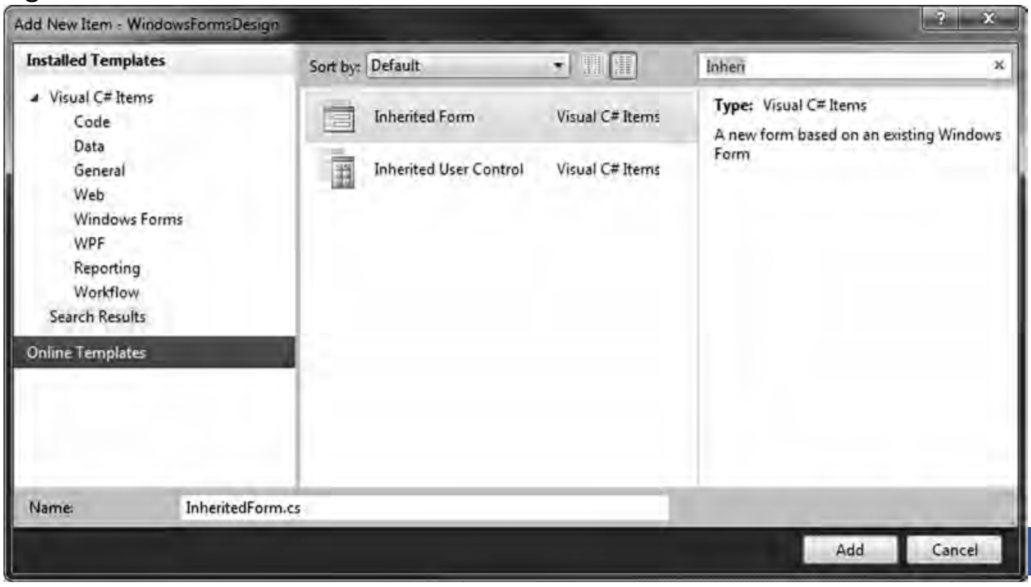
1. Open Form1.designer.cs and change the access modifiers for the label1 and dateTimePicker1 controls from private to protected, as shown:

   protected System.Windows.Forms.Label label1;

   protected System.Windows.Forms.DateTimePicker

   dateTimePicker1;
2. Select Project > Add Windows Forms to add a new Windows Form based on the Inherited Form template. You can quickly search for this template by typing its name in the search box, as shown in Figure 5-4. Name the inherited form InheritedForm.cs. (Note that the Inherited Form template is not available in Visual Studio Express edi- tions. If you are using an Express edition, just create a regular Windows Form named InheritedForm.cs and proceed to Step 4.)

Figure5-4


Add New Item - WindowsFormsDesign

Installed Templates
- Visual C# Items
  - Code
  - Data
  - General
  - Web
  - Windows Forms
  - WPF
  - Reporting
  - Workflow
- Search Results

Online Templates

Sort by: Default

Inherited Form	Visual C# Items
Inherited User Control	Visual C# Items

Inheri

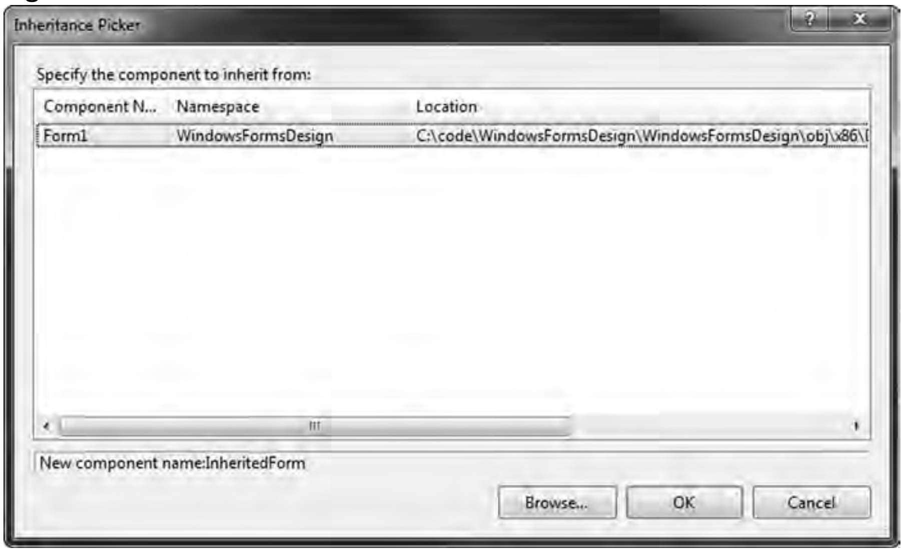
Type: Visual C# Items

A new form based on an existing Windows Form

Name: InheritedForm.cs

3. Click the Add button. Then, in the Inheritance Picker dialog box, select Form1 from the WindowsFormsDesign namespace, as shown in Figure 5-5, and click the OK button.

Figure 5-5


Inheritance Picker

Specify the component to inherit from:

Component N...	Namespace	Location
Form1	WindowsFormsDesign	C:\code\WindowsFormsDesign\WindowsFormsDesign\obj\x86\

New component name:InheritedForm

4. Select the Code view for the InheritedForm; you will see that the class InheritedForm inherits from Form1, as shown below. If you did not use the Inherited Form template in Step 2, you'll need to manually modify the code to add the code for inheritance (shown in bold):

```
public partial class InheritedForm
: WindowsFormsDesign.Form1
{
public InheritedForm()
{
InitializeComponent();
}
}
```

5. In the Designer view of the InheritedForm, set the Text property to Inherited Form.
6. Also in the Designer view, double-click InheritedForm. This action attaches an event handler for the Load event of the form and switches the view from Designer to Code. In Code view, change the default code for the event handler as follows:

```
private void InheritedForm_Load( object sender, EventArgs e){
label1.Text   =
dateTimePicker1.Value.ToLongDateString();
}
```

7. Open Program.cs and modify the Main method as shown below to make sure that InheritedForm is launched when you run the application:

```
[STAThread]
static void Main()
```

```
{
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Application.Run(new InheritedForm());
}
```

8. Select Debug > Start Debugging (or press F5) to run the project. When InheritedForm is loaded, the currently selected date is displayed on the Label control. This is unlike the previously created Form1, in which the Label control was initially empty.

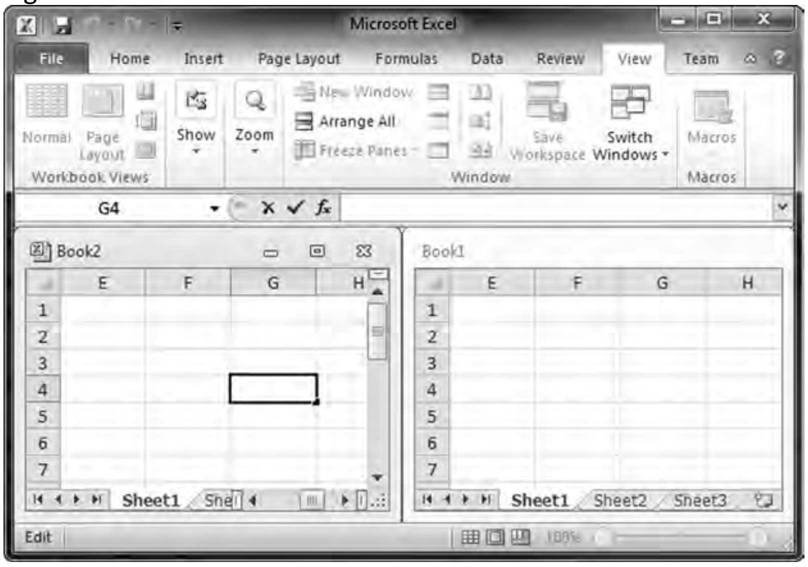
The InheritedForm form demonstrates that you can get all of Form1's functionality simply by inheriting the form. When you change the access modifier of Form1's member controls, label1 and dateTimePicker1, from private to protected, you will be able to access them from within the inherited form. This exercise also demonstrates how you can extend the functionality of the base form in an inherited form.

Understanding Multiple Document Interface (MDI) Applications

Multiple Document Interface (MDI) applications are applications in which multiple child windows reside under a single parent window.

MDI applications allow multiple windows to share a single application menu and toolbar. MDI applications often have a menu named Window that allows users to manage multiple child windows, offering features such as switching between child windows and arranging child windows. For example, Figure 5-6 shows Microsoft Excel 2010 in MDI mode.

Figure5-6



MDI applications contrast with single document interface (SDI) applications in which each window contains its own menu and toolbar. SDI applications rely on the operating system to provide window management functionality. For example, in Windows, you can switch among multiple windows by using the Windows Taskbar.

There is much debate among user-interface designers as to which application interface works best. Generally speaking, SDI is considered more suited to novice users, whereas MDI is con- sidered more suited to advanced users. Many popular applications such as Microsoft Word and Microsoft Excel support both SDI and MDI. Word and Excel install by default as SDI applications, but they provide users with an option to switch between SDI and MDI. For example, in Word 2010 and Excel 2010, you can switch to MDI mode by unchecking the "Show all windows in the Taskbar" option in the Options menu.

TakeNote

It can be tricky to implement support for multiple monitors in MDI applications because the parent window needs to span all of the user's monitors

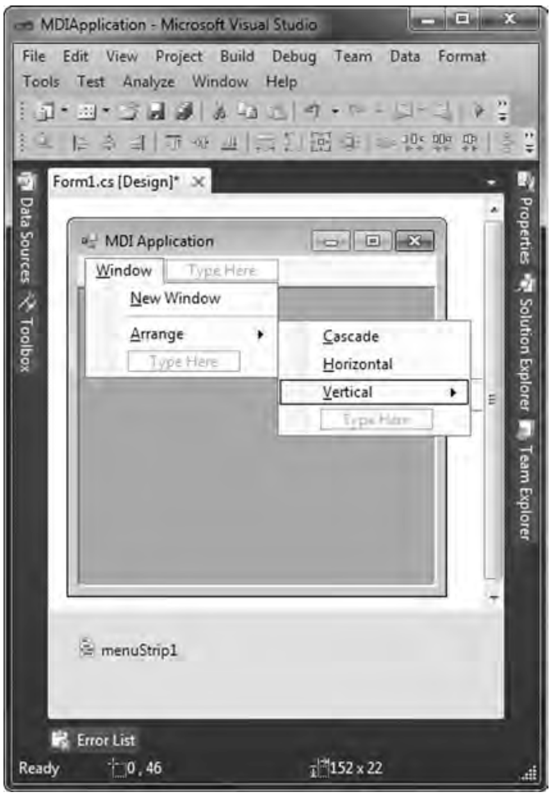
Illustration3: CREATE A MDI APPLICATION

GET READY. Launch Microsoft Visual Studio and create a new Windows Forms Application Project named MDIApplication. Then, perform these steps:

Select the Properties window for Form1 and set the Text property to MDI Application
and the IsMdiContainer property to True.

Select the MenuStrip control from the Toolbox and add it to the form. Add a top- level menu item &Window, and then add &New Window and &Arrange at the next level.

Under the Arrange menu, add three options—&Cascade, &Horizontal, and &Vertical—as shown in Figure 5-7
Figure 5-7



1. For the MenuStrip control, set its MdiWindowListItem property to the name of the Window menu (windowToolStripMenuItem by default).

2. In the Solutions Explorer, right-click the project and select Add > Windows Form.Add a new Windows Form with the name ChildForm.

3. Double-click the child form and add the following code to handle the Load event:

```
private void ChildForm_Load( object sender, EventArgs e)
{
Text           = DateTime.Now.ToString();
}
```

4. On the parent form, double-click the Window > New Window menu item and add the following event handler for its Click event:

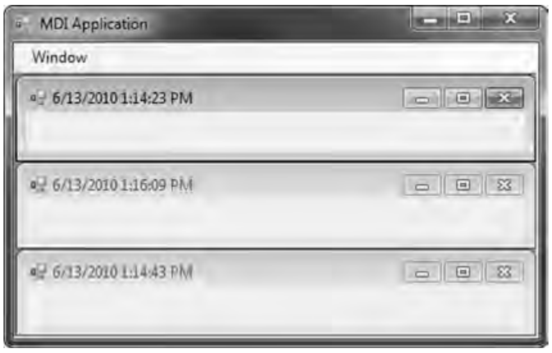
```
private void newWindowToolStripMenuItem_Click( object sender,
EventArgs e)
{
ChildForm child = new ChildForm();
 child.MdiParent = this;
child.Show();
}
```

5. On the parent form, double-click Window > Arrange, Window > Cascade, Window > Arrange, Window > Horizontal, and Window > Arrange, Window > Vertical, respectively, and add the following event handlers for their Click events:

```
private void cascadeToolStripMenuItem_Click( object sender,
EventArgs e)
{
LayoutMdi(MdiLayout.Cascade);
}
private void horizontalToolStripMenuItem_Click( object sender,
EventArgs e)
{
LayoutMdi(MdiLayout.TileHorizontal);
}
private void verticalToolStripMenuItem_Click( object sender,
EventArgs e)
{
LayoutMdi(MdiLayout.TileVertical);
}
```

6. Select Debug > Start Debugging (or press F5) to run the project. Select Window > New Window to create multiple new child windows. Switch among the child windows. Note that there is only one application instance in the Windows Taskbar. Now, use the options in the Window > Arrange menu to arrange the child windows. For example, an application with three child windows might look like the image in Figure 5-8 when the child windows are arranged horizontally

Figure5-8



Let’s review some of the important properties and methods used in this exercise. First, for the parent form, the IsMdiContainer property is set to true. This property indicates that the form is a container for multiple MDI child forms. Correspondingly, for each child form, you set the MdiParent property to specify the parent container form.

Next, the MdiWindowListItem property of the MenuStrip is used to indicate which menu item will be used to display the list of MDI child windows. When this property is set, the menu item will list all the child windows and also allow you to switch among child windows. As a result of the code in the ChildForm_Load method, the title bar for each form displays the date and time of the instant when the form was loaded.

Finally, the LayoutMdi method is used by the menu items in the Window menu to arrange the child windows. The method accepts a parameter of type MdiLayout enumeration. The enumeration value determines whether the child windows need to be tiled horizontally or vertically, cascaded, or displayed as icons

TakeNote

The & sign in front of a character in a menu’s text is not displayed as is; rather, it sets the character to be the shortcut key for the menu. For example, the &Window menu can be invoked by pressing Alt+W. The access keys will not be evident until the user presses the Alt key. A setting in the Windows operating system controls whether access keys are always visible

OBJECTIVE2: Understanding Console-Based Applications

Console-based applications do not have a graphical user interface and use a text-mode console window to interact with the user. These applications are best suited for tasks that require minimal or no user interface

As its name suggests, a console-based application is run from the console window. The input to this application can be provided using command-line parameters, or the application can interactively read characters from the console window. Similarly, the output of a console application is written to the command window as well. You can enable reading or writing to the console by creating an application using the Console Application template in Visual Studio

You can also use console-based applications to create commands that can be run from the command line. For example, you can take advantage of the pipes and filters provided by the operating system to pass the output of a command as input to another command, thereby creating more powerful commands by combining simple commands

TakeNote

To enable reading from or writing to the console from a Windows Forms application, set the project's Output Type to Console Application in the project's Properties

Working with Command-Line Parameters

In this section, you'll learn how to accept command-line parameters from a console application.

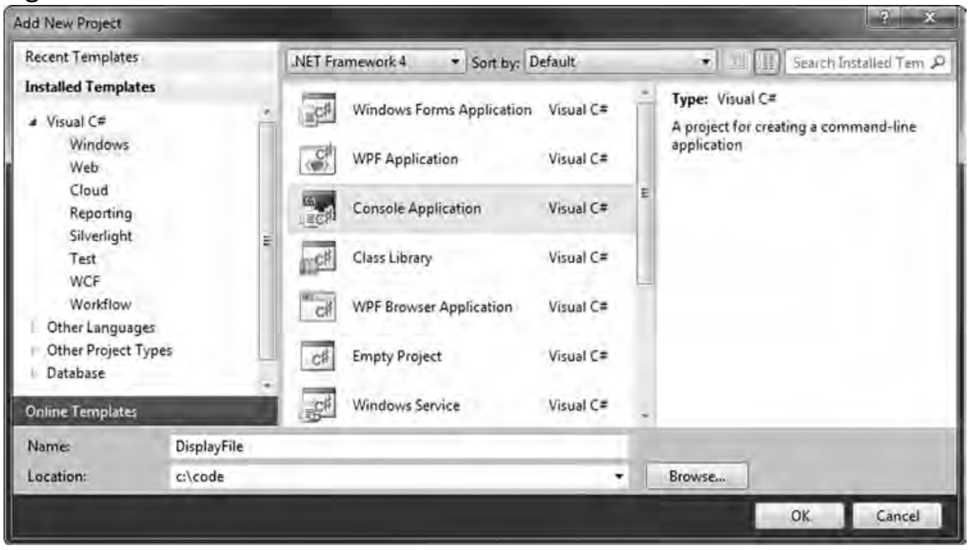
The following exercise creates a simple console application that accepts the name of a text file as a command-line argument and displays the contents of that file.

Illustration 4: CREATE A CONSOLE APPLICATION

GET READY. Launch Microsoft Visual Studio. Then, perform these steps:

1. Create a new project based on the Console Application template, as shown in Figure 5-9. Name the project DisplayFile.

Figure5-9



Type: Visual C#

A project for creating a command-line application

Name: DisplayFile

Location: c:\code

2. In the Program.cs, modify the code inside the Main method as shown below:

```
static void Main(string[] args)
{
if (args.Length < 1) return;
string[] lines = File.ReadAllLines(args[0]);
 foreach (string item in lines)
{
Console .Writeline(item);
}
}
```

3. Add the following using directive to the file:
   Using System.IO

4. Select Build > Build Solution (or press F6) to build the Project.

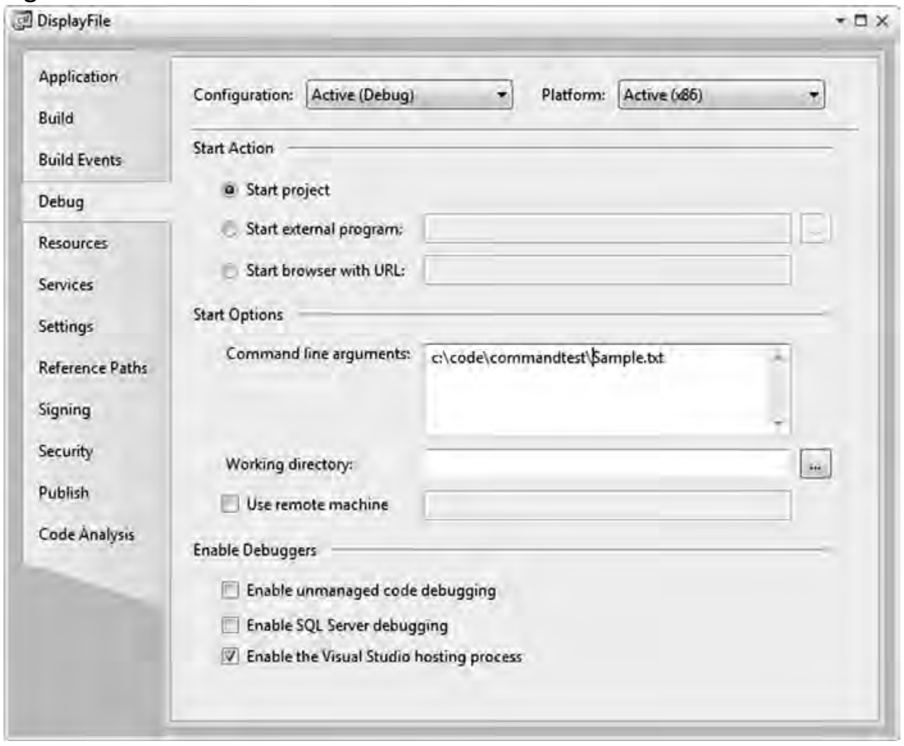
5. Create a text file using Visual Studio or Notepad, enter some sample text, and save the file as Sample.txt in the same folder as the executable file. (The executable file is created by default in the bin\debug folder under the project's folder.)

6. Open a command prompt and navigate to the path of the project's EXE file. Execute the following command:

   DisplayFile sample.txt
   This command should display the contents of the text file in the command window.

7. Alternatively, you can also pass the command line argument from within Visual Studio by using the Project's Properties window, as shown in Figure 5-10. To view the project's Properties window, select the Project > DisplayFile Properties menu option

Figure5-10



Configuration: Active (Debug) Platform: Active (x86)

Start Action
- Start project
- Start external program:
- Start browser with URL:

Start Options
- Command line arguments: c:\code\commandtest\Sample.txt
- Working directory:
- [ ] Use remote machine

Enable Debuggers
- [ ] Enable unmanaged code debugging
- [ ] Enable SQL Server debugging
- [✓] Enable the Visual Studio hosting process

OBJECTIVE3: Understanding Windows Services

A ***Windows service*** is an application that runs in the background and does not have any user interface.

The nature of Windows services make them ideal for creating long-running programs that run in the background and do not directly provide any user interaction. A Windows service can be started, paused, restarted, and stopped. A Windows service can also be set to start automatically when the computer is started.

Some examples of Windows services include a Web server that listens for incoming requests and sends a response, or an operating system's print spooler that provides printing services to the application programs

Services play an important role in enterprise application architecture. For example, you can have a service that listens for incoming orders and starts an order-processing workflow when- ever an order is received

TakeNote

Because a Windows service is capable of running in the background, it does not need a logged-on user in order to function. Windows services will run in their own Windows session in the specified security context. Still, depending on what permissions are needed, you can specify a user account under which to run the service

Creating a Windows Service

To create a Windows service in Visual Studio, use the Windows Service application tem- plate. Note that a Windows service must be installed before it can be used.

All Windows services must derive from the ServiceBase class. This base class provides the basic structure and functionality for creating a Windows service. You can override the base class methods OnStart, OnStop, OnPause, and OnContinue to add your own custom logic that executes in response to changes in service states.

The following exercise demonstrates how to create a simple Windows service that writes messages to the Application event log. Event logs are the part of Windows that is used by operating system tasks and applications running in the background to log error or other informational messages. Windows define three event logs by default: System, Application, and Security. Applications usually use the Application event log to log their message. The Windows Event Viewer utility can be used to view the messages in event logs.

TakeNote

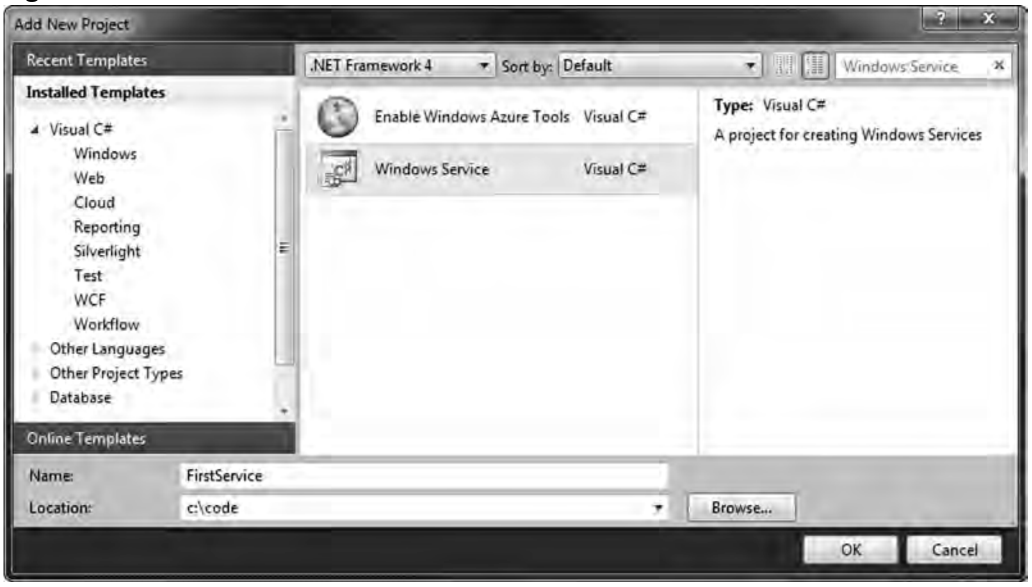
Visual Studio Express Edition does not provides templates for creating Windows service projects. Thus, you will need a non-express version of Visual Studio to complete exercises that use Windows service projects

Illustration5: Create a Window Service

GET READY. Launch Microsoft Visual Studio. Then, perform these steps:

1. Create a new project based on the Windows Service template. Name the project FirstService, as shown in Figure 5-11.

Figure5-11



Type: Visual C#

A project for creating Windows Services

Name: FirstService

Location: c:\code

2. Select the Properties window for Service1 and set the (Name) and ServiceName properties to "FirstService".
3. In the Solution Explorer, rename the file Service1.cs as FirstService.cs. Open Program.cs and verify that the references to Service1 have been changed to FirstService.
4. Select the Properties window for the service and set the CanPauseAndContinue property and the CanShutdown property to True.
5. Open the designer for FirstService and add an EventLog component to it from the Toolbox. The EventLog component allows you to connect with the event logs.

6. View the code for FirstService and modify the constructor as shown below. In this code, you first create an event source by the name FirstService. This event source is used to distinguish messages generated by a specific application from all other messages in an event log. Then, you set the Source property of the event log component to the name of the event source. The Log property of the event log component, eventLog1, is used to specify the event log used to record the messages:

```
public FirstService()
{
InitializeComponent();
if    (!EventLog.SourceExists("FirstService"))
{
EventLog.CreateEventSource( "FirstService", "Application");
}
eventLog1.Source    = "FirstService"; eventLog1.Log  = "Application";
}
```

7. Add the following code to the service state change methods to define their behavior.

The WriteEntry method of the event log component, eventLog1, is used to write a message to an event log. As part of the method, you can specify the type of message. For example, your message can be an error message, a warning message, or just a piece of information:

```
protected override void OnStart(string[] args)
{
eventLog1.WriteEntry( “Starting        the service”,
EventLogEntryType.Information,        1001);
}

protected override void OnStop()
{
eventLog1.WriteEntry( “Stopping        the service”,
EventLogEntryType.Information,        1001);
}
protected override void OnPause()
{
eventLog1.WriteEntry( “Pausing         the service”,
EventLogEntryType.Information,        1001);
}
protected override void OnContinue()
{
eventLog1.WriteEntry( “Continuing      the service”,
EventLogEntryType.Information,        1001);
}
protected override void OnShutdown()
{
eventLog1.WriteEntry(
“Shutting     down the service”, EventLogEntryType.Information,
     1001);
}
```

8. Select **Build > Build Solution** (or press **F6**) to build the project

Here, the code for FirstService overrides the OnStart, OnStop, OnPause, OnContinue, and OnShutdown methods to write messages to the event log. Not all services need to override these methods, however. Whether a service needs to override these methods depends on the value of the CanPauseAndContinue and CanShutdown properties of the Windows service.

The eventlog’s WriteEntry method takes the message to write to the log, the type of event log entry (information, error, warning, etc.), and an eventId, which is an application-specific id used to identify the event.

The FirstService Windows service is now ready, but before it can be used, it must be installed in the Windows service database. This is done by adding a Service Installer to the Windows service project. The following exercise shows how to do this.

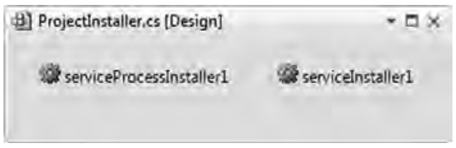
Illustration6: ADD AN INSTALLER TO WINDOW SERVICE

GET READY. Launch Microsoft Visual Studio. Then, perform these steps:

1. Open the FirstService project created in the previous exercise. Right-click the Designer surface of FirstService.cs and select the Add Installer option from the context menu.
2. This action adds a new file ProjectInstaller.cs to the project. Open the Designer for

ProjectInstaller.cs. You should see that two components were added to the Designer, as shown in Figure 5-12.

Figure 5-12



3. Access the properties for the serviceProcessInstaller1component, and change the Account property to LocalService.
4. Next, access the properties for the serviceInstaller1component. Change the DisplayName property to FirstService and the Description property to “A simple test service.” Note that the value of the StartType property is set by default to Manual.
5. Select Build > Build Solution (or press F6) to build the project. The Windows service is now ready to be installed

TakeNote

The StartType property of the ServiceInstaller class indicates how and when a service is started. The StartType property can have one of three possible values. The value Manual, which is also the default value, indicates that you need to start the service manually. The value Automatic indicates that the service will be started automatically when Windows is started. The value Disabled indicates that the service cannot be started.

To minimize security risks, you should refrain from using the LocalSystem account for running a Windows service unless that service requires higher security privileges to function

When you add an installer to a Windows Service project, the ServiceProcessInstaller and the ServiceInstaller classes are added to the project. The ServiceProcessInstaller class performs installation tasks that are common to all the Windows services in an application. This includes setting the login account for the Windows service. The ServiceInstaller class, on the other hand, performs the installation tasks that are specific to a single Windows service, such as setting the ServiceName and StartType.

The Account property of the ServiceProcessInstaller class specifies the type of account under which the services run. The Account property is of the type ServiceAccount enumeration where the possible values are LocalService, LocalSystem, NetworkService, and User. The LocalSystem value specifies a highly privileged account, whereas the LocalService account acts as a nonprivileged user.

An executable file that has the code for the service installer classes can be installed by using the command line Installer tool (installutil.exe). The following exercise shows how to install a Windows service application in the Windows service database.

Illustration 7: INSTALL A WINDOW SERVICE

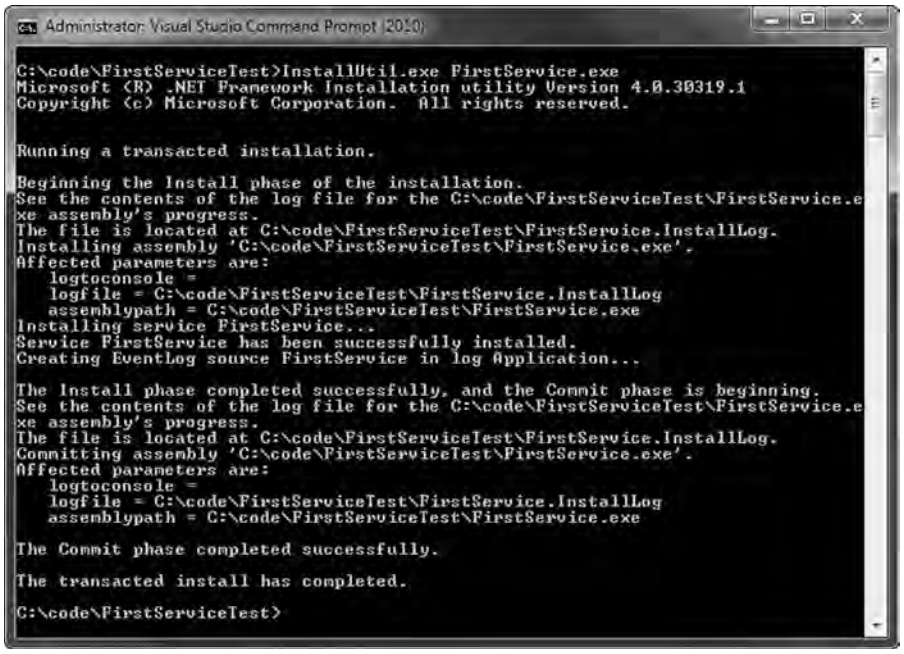
GET READY. To install a Windows service, take the following steps:

1. Run Visual Studio Command Prompt as administrator. To access the command prompt, go to Start > All Programs > Visual Studio > Visual Studio Tools, then choose Visual Studio Command Prompt. To run a program as administrator in Windows, right-click on the program shortcut and select the Run as administrator option from the shortcut menu.

2. Change the directory to the output directory of the FirstService project. This is the directory where the executable file is located

3. Issue the following command; you should see results like those shown in Figure 5-13
installutil FirstService.exe

Figure 5-13



```
C:\code\FirstServiceTest>InstallUtil.exe FirstService.exe
Microsoft (R) .NET Framework Installation utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.


Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the C:\code\FirstServiceTest\FirstService.e
xe assembly's progress.
The file is located at C:\code\FirstServiceTest\FirstService.InstallLog.
Installing assembly 'C:\code\FirstServiceTest\FirstService.exe'.
Affected parameters are:
   logtoconsole =
   logfile = C:\code\FirstServiceTest\FirstService.InstallLog
   assemblypath = C:\code\FirstServiceTest\FirstService.exe
Installing service FirstService...
Service FirstService has been successfully installed.
Creating EventLog source FirstService in log Application...

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the C:\code\FirstServiceTest\FirstService.e
xe assembly's progress.
The file is located at C:\code\FirstServiceTest\FirstService.InstallLog.
Committing assembly 'C:\code\FirstServiceTest\FirstService.exe'.
Affected parameters are:
   logtoconsole =
   logfile = C:\code\FirstServiceTest\FirstService.InstallLog
   assemblypath = C:\code\FirstServiceTest\FirstService.exe

The Commit phase completed successfully.

The transacted install has completed.

C:\code\FirstServiceTest>
```

4. The Windows service FirstService is now installed

The Windows service application is now stored in the Windows service database. Earlier, when you added a ServiceInstaller for the FirstService, you set the StartType property of the serviceInstaller1 component to Manual. As a result, you'll need to manually start the service when needed. The following exercise demonstrates how to start, pause, continue, and stop a Windows service.

TakeNote

Installing a Windows service requires access to Windows Registry. Therefore, be sure to run installUtil.exe as an administrator

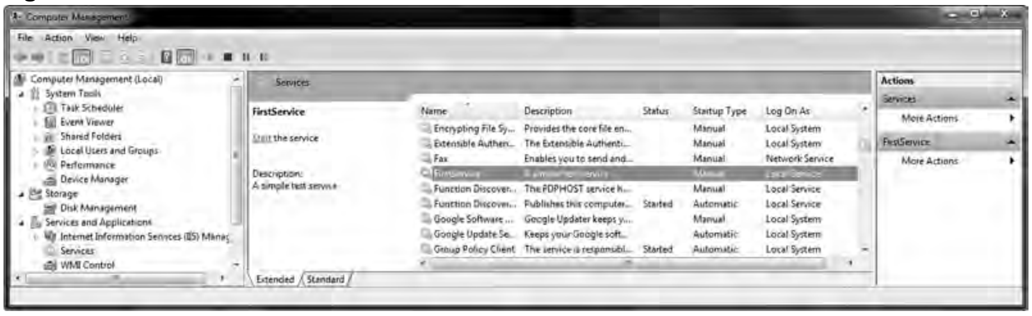
To uninstall a Windows service, use InstallUtil. exe with the option -u.

Illustration 8: WORK WITH A WINDOW SERVICE

GET READY. Launch the Computer Management window by right-clicking My Computer and selecting Manage from the shortcut menu. Then, perform these steps

1. In the Computer Management window, expand the Services and Applications section and select Services. A list of all services installed on the computer should be dis- played, as shown in Figure 5-14.

Figure5-14



FirstService

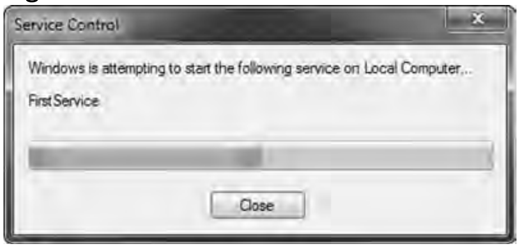
Start the service

Description:
A simple test service

Name	Description	Status	Startup Type	Log On As
Encrypting File Sy...	Provides the core file en...		Manual	Local System
Extensible Authen...	The Extensible Authenti...		Manual	Local System
Fax	Enables you to send and...		Manual	Network Service
FirstService	A simple test service		Manual	Local Service
Function Discover...	The FDPHOST service h...		Manual	Local Service
Function Discover...	Publishes this computer...	Started	Automatic	Local Service
Google Software ...	Google Updater keeps y...		Manual	Local System
Google Update Se...	Keeps your Google soft...		Automatic	Local System
Group Policy Client	The service is responsibl...	Started	Automatic	Local System

2. Select the FirstService service and click on the Start hyperlink, as shown in Figure 5-14.You should see a dialog box indicating progress, as shown in Figure 5-15. When the ser- vice is started, the status of the service will change to Started.

Figure 5-15

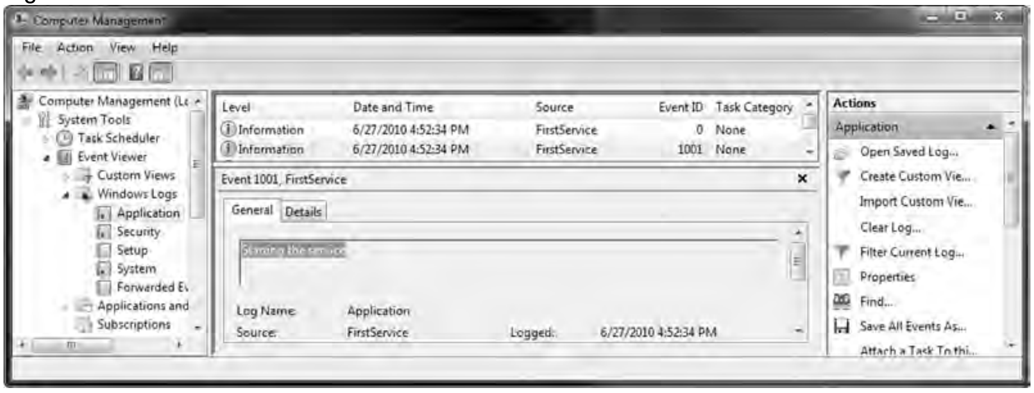


Windows is attempting to start the following service on Local Computer...

First Service

3. Expand the **Event Viewer** node and select the **Application Windows** log. You should see a message from FirstService that says "Starting the Service," as shown in Figure 5-16

Figure 5-16



Level	Date and Time	Source	Event ID	Task Category
Information	6/27/2010 4:52:34 PM	FirstService	0	None
Information	6/27/2010 4:52:34 PM	FirstService	1001	None

Event 1001, FirstService

Starting the service

Log Name: Application
Source: FirstService Logged: 6/27/2010 4:52:34 PM

4. Go back to the list of Services and attempt to pause, resume, or stop FirstService.Check the Application event log to verify that the appropriate messages are being displayed